Utilisation de la librairie Lib_Temp_PIC24FJ256GA702_V2-0

IUT GEII MARSEILLE
2020 - 2021

SOMMAIRE

LE MICROCONTROLEUR

1 - Aspect matériel

Le microcontrôleur est constitué d'un CPU (Central Processor Unit) qui exécute le programme et de plusieurs périphériques ou module comme les GPIO (General Purpose Input Output), l'UART (Universal Asynchronous Receiver Transmitter) et biens d'autres.

1.1 - Liste des périphériques

Voici une liste proposée par Microchip :

TABLE 1: PIC24FJ256GA705 FAMILY DEVICES

Device	Memory		Pins	GPIO	DMA Channels	Peripherals													JTAG
	Program (bytes)	SRAM (bytes)				10/12-Bit A/D Channels	Comparators	CRC	MCCP 6-Output/2-Output	IC/OC/PWM	16-Bit Timers	I²C	Variable Width SPI	LIN-USART/IrDA®	CTMU Channels	EPMP (Address/Data Line)	CLC	RTCC	
PIC24FJ64GA705	64K	16K	48	40	6	14	3	Yes	1/3	3/3	3	2	3	2	13	10/8	2	Yes	Yes
PIC24FJ128GA705	128K	16K	48	40	6	14	3	Yes	1/3	3/3	3	2	3	2	13	10/8	2	Yes	Yes
PIC24FJ256GA705	256K	16K	48	40	6	14	3	Yes	1/3	3/3	3	2	3	2	13	10/8	2	Yes	Yes
PIC24FJ64GA704	64K	16K	44	36	6	14	3	Yes	1/3	3/3	3	2	3	2	13	10/8	2	Yes	Yes
PIC24FJ128GA704	128K	16K	44	36	6	14	3	Yes	1/3	3/3	3	2	3	2	13	10/8	2	Yes	Yes
PIC24FJ256GA704	256K	16K	44	36	6	14	3	Yes	1/3	3/3	3	2	3	2	13	10/8	2	Yes	Yes
PIC24FJ64GA702	64K	16K	28	22	6	10	3	Yes	1/3	3/3	3	2	3	2	12	No	2	Yes	Yes
PIC24FJ128GA702	128K	16K	28	22	6	10	3	Yes	1/3	3/3	3	2	3	2	12	No	2	Yes	Yes
PIC24FJ256GA702	256K	16K	28	22	6	10	3	Yes	1/3	3/3	3	2	3	2	12	No	2	Yes	Yes

Tous ces périphériques sont contrôlés par des registres qui se trouvent dans la RAM.
Certains sont aussi configurés au démarrage du microcontrôleur par les bits de configurations.

La plupart de ces périphériques possèdent 2 bits d'activation :

- 1 bit "Enable" qu'on retrouve dans les registres sous la forme **xxxEN**
- 1 bit "Peripheral Module Disable" (PMD) sous la forme **xxxMD**.

Les 2 bits ont des fonctions similaires, c'est à dire activer ou désactiver le périphérique, sauf que le 2e, celui identifié par **xxxMD**, permet de couper toutes les horloges du périphérique dans un soucis d'économie d'énergie.

TABLE 10-2: PERIPHERAL MODULE DISABLE REGISTER SUMMARY

Register	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
PMD1	—	—	T3MD	T2MD	T1MD	—	—	—	I2C1MD	U2MD	U1MD	SPI2MD	SPI1MD	—	—	ADCMD	0000
PMD2	—	—	—	—	—	IC3MD	IC2MD	IC1MD	—	—	—	—	—	OC3MD	OC2MD	OC1MD	0000
PMD3	—	—	—	—	—	CMPMD	RTCCMD	PMPMD	CRCMD	—	—	—	—	—	I2C2MD	—	0000
PMD4	—	—	—	—	—	—	—	—	—	—	—	—	REFOMD	CTMUMD	LVDMD	—	0000
PMD5	—	—	—	—	—	—	—	—	—	—	—	—	CCP4MD	CCP3MD	CCP2MD	CCP1MD	0000
PMD6	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	SPI3MD	0000
PMD7	—	—	—	—	—	—	—	—	—	—	DMA1MD	DMA0MD	—	—	—	—	0000
PMD8	—	—	—	—	—	—	—	—	—	—	—	—	CLC2MD	CLC1MD	—	—	0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

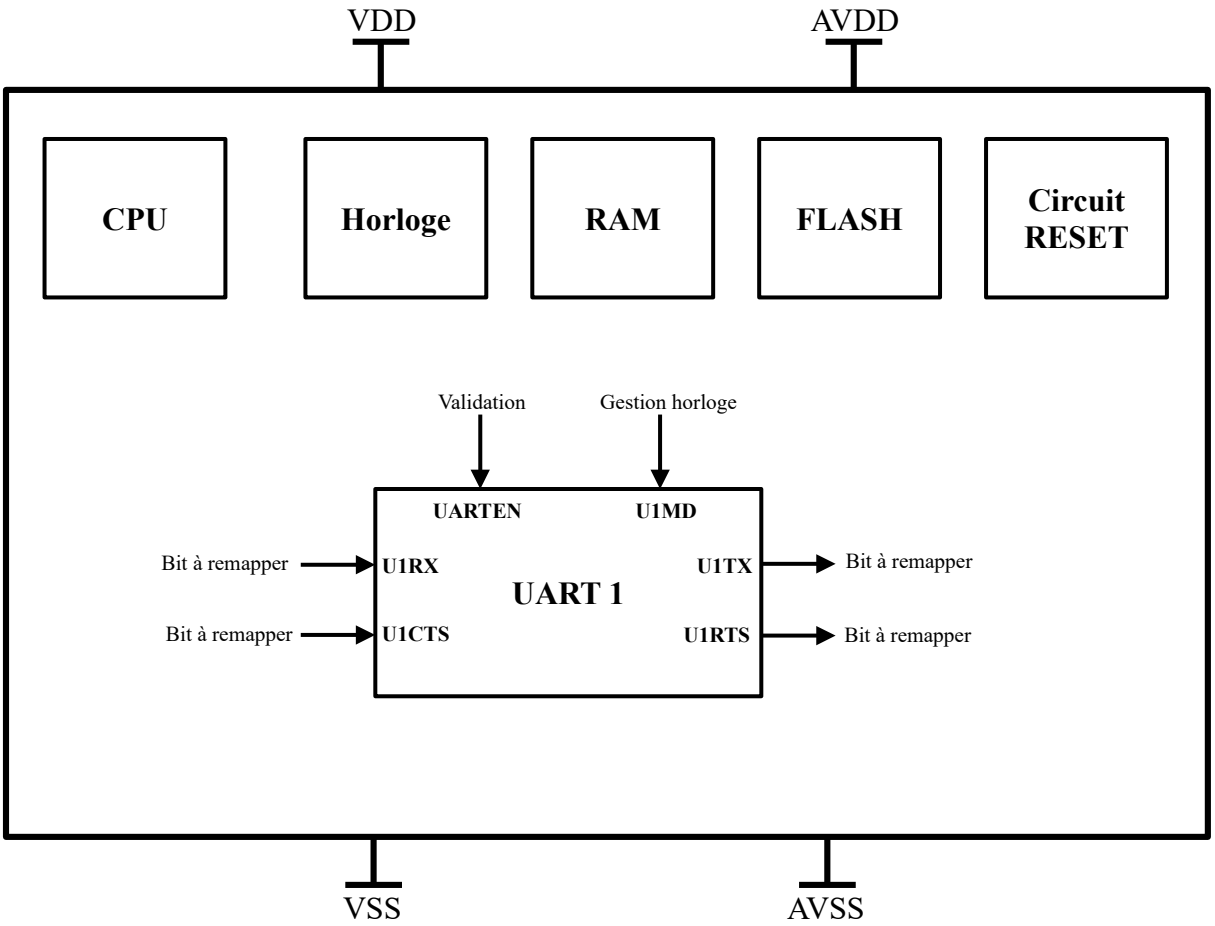
1.2 - Architecture interne simplifiée du microcontrôleur

Les liaisons entre les différents périphériques ne sont pas faites pour ne pas alourdir le schéma.
Mais chacun des périphériques représentés ci-dessous possède :

- un bus d'adresse
- un bus de données
- un bus de contrôle
- et des bornes d'alimentations (**VDD** ou **AVDD** et **VSS**) à l'intérieur du microcontrôleur.

Tous ces signaux n'apparaissent pas sur le schéma suivant.

Dans l'exemple qui suit on va utiliser le périphérique **UART1** avec la librairie proposée :



Sur ce schéma les 5 modules : **CPU**, **Horloge**, **RAM**, **FLASH** et **Circuit Reset** seront toujours présents pour l'utilisation de n'importe quel périphérique du microcontrôleur.

On remarque que **UART1** possède 4 bits à remapper : **U1RX**, **U1TX**, **U1CTS** et **U1RTS**. Ces 4 bits existent dans le microcontrôleur mais ne sont pas connectés sur les bornes physiques externes.
Dans notre application on utilisera uniquement les bits **U1RX** et **U1TX**. Les 2 autres bits ne seront pas utilisés. Il faut donc qu'on connecte **U1RX** et **U1TX** sur des bornes externes, par exemple :

- **U1RX** sur **RB7**
- **U1TX** sur **RB6**.

D'après la Datasheet du microcontrôleur page 4, **RB6** porte aussi le nom de **RP6** et **RB7** porte aussi le nom de **RP7**.
Lorsque une borne externe porte le nom **RPx** (x représente un numéro) alors celle ci peut être connectée avec les signaux internes disponibles dans le microcontrôleur.

Pour faire le remappage il faut lire le paragraphe "**11.5 Peripheral Pin Select (PPS)**" de la Datasheet du microcontrôleur PIC24FJ256GA702.

Autre solution on utilise la librairie "**Lib_Temp_PIC24FJ256GA702_V1-4.h**" et en particulier les fonctions dont les prototypes sont :

- void **Lib_Temp_REMAP_RPx_IN**(unsigned char **Nom_Signal_In**, unsigned char **Numero_RP**);
- void **Lib_Temp_REMAP_RPx_OUT**(unsigned char **Nom_Signal_Out**, unsigned char **Numero_RP**);

Dans cette librairie on trouve aussi les variables symboliques suivantes :

- #define **REMAP_SIGNAL_IN_U1RX** 19
- #define **REMAP_SIGNAL_OUT_U1TX** 3

- #define **RP6_RB6_BORNE_15** 6
- #define **RP7_RB7_BORNE_16** 7

En ce qui concerne le signal **U1RX**, c'est un signal d'entrée, il doit donc être remappé avec la fonction : **Lib_Temp_REMAP_RPx_IN**.

Pour le signal **U1TX**, c'est un signal de sortie donc remappé avec **Lib_Temp_REMAP_RPx_OUT**.

Ce qui nous amène à l'écriture suivante en langage C :

```
Lib_Temp_REMAP_RPx_IN(REMAP_SIGNAL_IN_U1RX, RP7_RB7_BORNE_16);
Lib_Temp_REMAP_RPx_OUT(REMAP_SIGNAL_OUT_U1TX, RP6_RB6_BORNE_15);
```

Ensuite on doit configurer l'**UART1** en utilisant les fonctions suivantes :

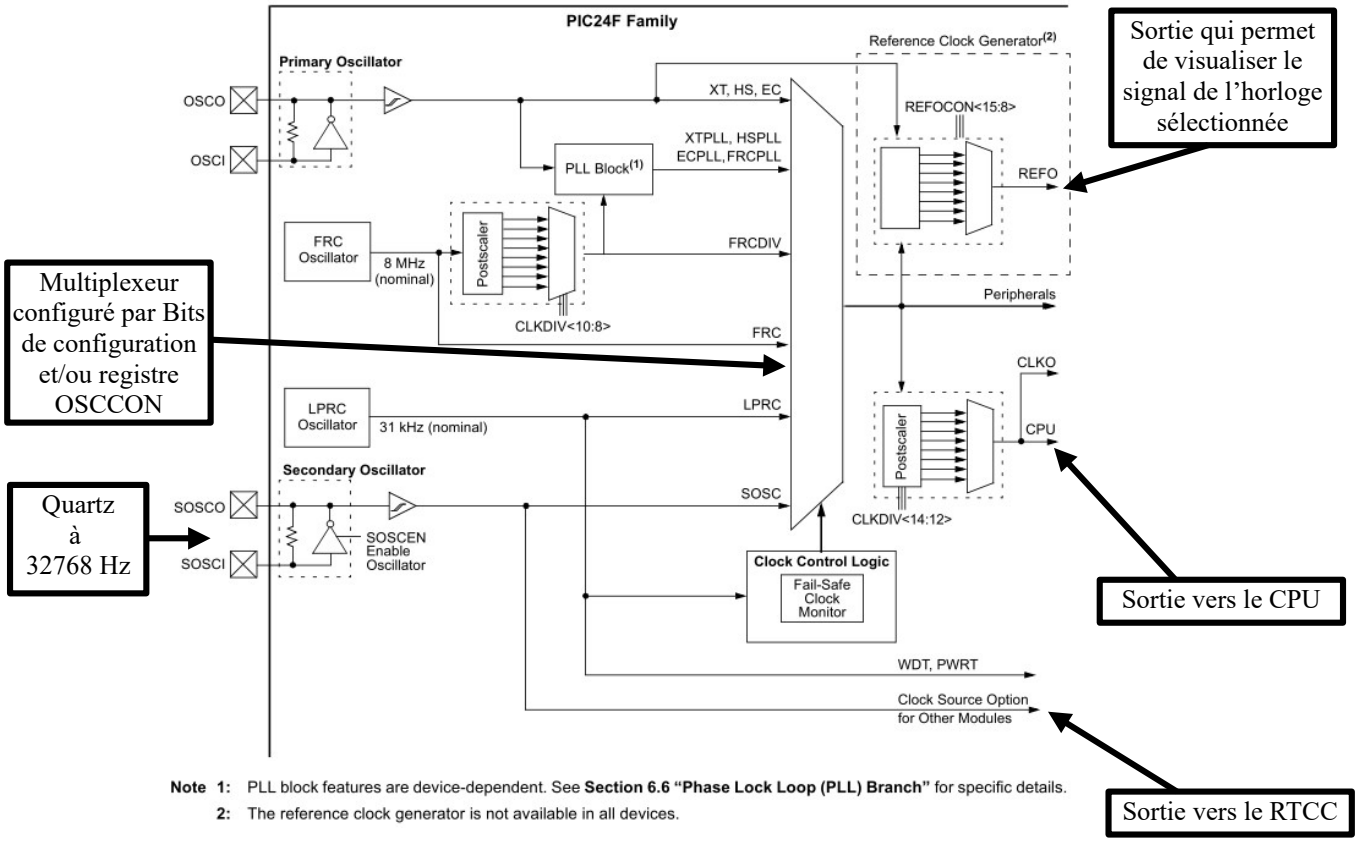
```
void Lib_Temp_UART_Config(unsigned char Numero_UART, float Fcyc, float Vitesse_en_Baud);
float Lib_Temp_UART_Vitesse_de_Transmission(float Fcyc, float Vitesse_en_Baud);
void Lib_Temp_UART_Activation_Interruption_RX(unsigned char On_Off, unsigned char Niveau_Priorite);
void Lib_Temp_UART_Activation_Interruption_TX(unsigned char On_Off, unsigned char Niveau_Priorite);
```

La fonction **Lib_Temp_UART_Config** d'activer les horloges du module en activant le bit **U1MD** et permet aussi de configurer l'UART avec le protocole de communication suivant : 8 bits de données, pas de parité, 1 bit de stop et pas de contrôle de flux de données.
Il faudra ensuite préciser le numéro de l'UART utilisée (1 ou 2) par la variable "**Numero_UART**" et la vitesse de transmission grâce à la variable "**Vitesse_en_Baud**".

La fonction **Lib_Temp_UART_Vitesse_de_Transmission** permet de changer la vitesse de transmission.

Les fonctions **Lib_Temp_UART_Activation_Interruption_RX** et **Lib_Temp_UART_Activation_Interruption_TX** permettent de gérer l'activation des interruptions pour la réception et l'émission.

2 - Etude des horloges



Pour notre application on utilisera 2 horloges :

- une horloge pour faire fonctionner le CPU (horloge FRC)
- une horloge pour faire fonctionner le périphérique RTCC (horloge secondaire SOSC).

Les "bits de configurations" permettent de configurer l'ensemble du microcontrôleur avant la mise sous tension de ce dernier. Ils permettent entre autre de sélectionner une horloge par défaut pour le CPU au moment du démarrage du programme.
Par exemple avec la configuration suivante :
#pragma config FNOSC = FRC // Oscillator Source Selection (Internal Fast RC (FRC))
Le microcontrôleur démarrera avec l'horloge interne FRC.

Ensuite il faut configurer le registre **OSCCON** et suivants pour gérer tous les paramètres des horloges (voir le paragraphe : "**9.0 Oscillator Configuration**" de la Datasheet).

Il existe un module, nommé **REFO** (REference Clock Output) qui permet de sortir l'une des horloges internes du microcontrôleur sur une borne externe de la même manière qu'on a fait ci-dessus avec le signal **U1TX**. Le signal de sortie de ce module se nomme **REFO**. Pour activer et sélectionner une des horloges présente il faut configurer les registres **REFOCONL** et **REFOCONH**, sans oublier de remapper le signal **REFO** sur une borne de sortie du microcontrôleur.

3 - Aspect logiciel

Dans cette partie on ne configurera que l'**UART 1** et on ne fera apparaître que le code partiel de votre programme à réaliser.

3.1 - Configuration de l'UART 1

Après avoir créé les fichiers "**UART.c**" et "**UART.h**", il faudra écrire le code suivant :

3.1.1 - Fichier : Main.h

On le complète avec la ligne :
#include "UART.h"

3.1.2 - Fichier : UART.h

```
#define HORL_FCY          4e6              // Définit une valeur pour la fréquence Fcy

unsigned char UART_Carac_Recu;             // Variable globale qui contient le caractère reçu
unsigned char UART_Indique_Reception;      // Variable globale qui indique la réception d'un caractère
void UART_Init(void) ;
```

3.1.3 - Fichier : UART.c

```
#include "Main.h"

//-----------------------------------------------------------------------------
// Initialisation du module UART
//-----------------------------------------------------------------------------
void UART_Init(void)
{
   Lib_Temp_REMAP_SIGNAL_IN_SUR_RPx(REMAP_SIGNAL_IN_U1RX, RP7_RB7_BORNE_16);
   Lib_Temp_REMAP_SIGNAL_OUT_SUR_RPx(REMAP_SIGNAL_OUT_U1TX, RP6_RB6_BORNE_15);

   Lib_Temp_UART_Config(1, (float)HORL_FCY, 115200);
   Lib_Temp_UART_Activation_Interruption_RX(ON, 3);
   Lib_Temp_UART_Activation_Interruption_TX(OFF, 3);
}
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Interruption U1RX - Réception UART 1
//-----------------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _U1RXInterrupt(void)
{
   UART_Carac_Recu          = U1RXREG;        // Lecture du caractère reçu
   UART_Indique_Reception   = 1;              // Signale la réception d'un caractère lorsqu'elle est à 1
   IFS0bits.U1RXIF          = 0;              // Init du bit de réception
}
//-----------------------------------------------------------------------------
```

3.1.4 - Fichier : Main.c

```
#include "../Sources/Main.h"
#include "../Sources/Config.h"

//-----------------------------------------------------------------------
// Fonction principale
//-----------------------------------------------------------------------
int main(void)
{
   MAIN_Init();


   while(1)
   {

   }
}
//-----------------------------------------------------------------------


//-----------------------------------------------------------------------
// Initialisation principale
//-----------------------------------------------------------------------
void MAIN_Init(void)
{
   HORLOGE_Init();          // Initialisation des horloges

   UART_Init();             // Initialisation de l'UART 1
}
//-----------------------------------------------------------------------
```

LES FONCTIONS DE LA LIBRAIRIE

1 - L'ADC

```
// A utiliser avec la variable : unsigned char ADC_Anx de la fonction  Lib_Temp_ADC_Config_ANx
#define ADC_AN0_RA0_BORNE_2         26
#define ADC_AN1_RA1_BORNE_3         27
#define ADC_AN2_RB0_BORNE_4         0
#define ADC_AN3_RB1_BORNE_5         1
#define ADC_AN4_RB2_BORNE_6         2
#define ADC_AN5_RB3_BORNE_7         3
#define ADC_AN6_RB14_BORNE_25       14
#define ADC_AN7_RB13_BORNE_24       13
#define ADC_AN8_RB12_BORNE_23       12
#define ADC_AN9_RB15_BORNE_26       15

#define BIT_RA0         26
#define BIT_RA1         27
#define BIT_RA2         28
#define BIT_RA3         29
#define BIT_RA4         30
#define BIT_RB0         0
#define BIT_RB1         1
#define BIT_RB2         2
#define BIT_RB3         3
#define BIT_RB4         4
#define BIT_RB5         5
#define BIT_RB6         6
#define BIT_RB7         7
#define BIT_RB8         8
#define BIT_RB9         9
#define BIT_RB10        10
#define BIT_RB11        11
#define BIT_RB12        12
#define BIT_RB13        13
#define BIT_RB14        14
#define BIT_RB15        15


// A utiliser avec la fonction : Lib_Temp_ADC_Config_ANx
typedef struct
{
   unsigned char   Bit_Voie_de_Conversion_ADC_ANx;       /*ADC_AN0_RA0_BORNE_2 à ADC_AN9_RB15_BORNE_26*/
   unsigned char   Bit_Alimentation_Capteur_Temperature; /*BIT_RA0 à BIT_RB15*/
   unsigned char   Autorise_Interruption;                /*0:non autorisé  -  1:autorisé*/
   unsigned char   Niveau_Priorite;                      /*Niveau de 0 à 7*/
}ADC_CONFIG;

void Lib_Temp_ADC_Config_ANx(ADC_CONFIG Config_ADC);       // Converti 16 mesures sur la voie ADC_ANx
void Lib_Temp_ADC_Alimente_Capteur_Temperature(unsigned char On_Off);
void Lib_Temp_ADC_Activation_Interruption(unsigned char On_Off, unsigned char Niveau_Priorite);
void Lib_Temp_ADC_Start_Conversion_Automatique(void);
void Lib_Temp_ADC_Stop_Conversion_Automatique(void);
char Lib_Temp_ADC_Attend_Fin_Conversion(void);         // retourne 1 si fin de conversion sinon retourne -1 si probleme
unsigned int Lib_Temp_ADC_Resultat_Conversion_Somme_des_8_derniers_Echantillons(void);
                                                      // retourne la somme des 8 derniers buffers de conversions
unsigned int Lib_Temp_ADC_Temperature_en_Centieme_de_degre(unsigned int V_Alim_en_mV);
                                                      // retourne les 8 derniers buffers de conversions en temperature


//-------------------------------------------------------------------------
// Interruption de l'ADC1
//-------------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _ADC1Interrupt(void)
{

   AD1CON1bits.ASAM     = 0;          // Stop auto conversion
   IFS0bits.AD1IF       = 0;
}
//-------------------------------------------------------------------------
```

La fonction "**Lib_Temp_ADC_Config_ANx**" associée à la variable de type "**ADC_CONFIG**" permet la configuration de l'ADC.

La fonction "**Lib_Temp_ADC_Alimente_Capteur_Temperature**" permet d'alimenter le capteur de température.

La fonction "**Lib_Temp_ADC_Activation_Interruption**" permet d'activer l'interruption de l'ADC.

Les fonctions "**Lib_Temp_ADC_Start_Conversion_Automatique**" et "**Lib_Temp_ADC_Stop_Conversion_Automatique**" permettent de lancer ou d'arrêter la conversion automatique de l'ADC.

La fonction "**Lib_Temp_ADC_Attend_Fin_Conversion**" permet d'attendre la fin de conversion de l'ADC lorsqu'il est utilisé en mode scrutation.

La fonction "**Lib_Temp_ADC_Temperature_en_Centieme_de_degre**" calcule la température après la conversion et retourne le résultat en centième de degré.

2 - Le RTCC

Ce périphérique gère un calendrier.

Listes des fonctions disponibles :

```
typedef struct
{
   unsigned char Second;
   unsigned char Minute;
   unsigned char Heure;
   unsigned char JourSemaine;      // 0 pour Dimanche
   unsigned char Jour;
   unsigned char Mois;
   unsigned char An;
} RTCC_DATE;

void Lib_Temp_RTCC_Config(void);
void Lib_Temp_RTCC_Ecriture_Date_Dans_Module_RTCC_du_Microcontroleur(RTCC_DATE Date);
void Lib_Temp_RTCC_Lecture_Date(RTCC_DATE *Date);
void Lib_Temp_RTCC_Activation_Sortie_RTCC_PPS(unsigned char On_Off);
void Lib_Temp_RTCC_Activation_Interruption(unsigned char On_Off, unsigned char Niveau_Priorite);
void Lib_Temp_RTCC_Conversion_Date_En_Seconde(RTCC_DATE Date, unsigned long *Date_en_Seconde);
void Lib_Temp_RTCC_Conversion_Seconde_En_Date(unsigned long Date_en_Seconde, RTCC_DATE *Date);
void Lib_Temp_RTCC_Conversion_Seconde_en_Date_String(unsigned long Date_en_Seconde, char *Date_string);
void Lib_Temp_RTCC_Conversion_Date_en_Date_String(RTCC_DATE Date, char *Date_string);


//-----------------------------------------------------------------------------
// Interruption RTCC
//-----------------------------------------------------------------------------
void __attribute__((interrupt, auto_psv)) _RTCCInterrupt(void)
{
   RTCCON1Hbits.ALRMEN        = 1;
   IFS3bits.RTCIF             = 0;          // Init du bit de reception
}
//-----------------------------------------------------------------------------
```

La fonction "**Lib_Temp_RTCC_Config**" permet d'activer le module RTCC.

La fonction "**Lib_Temp_RTCC_Ecriture_Date_Dans_Module_RTCC_du_Microcontroleur**" permet de changer la date du module RTCC.

La fonction "**Lib_Temp_RTCC_Lecture_Date**" permet de lire la date du module RTCC.

La fonction "**Lib_Temp_RTCC_Activation_Interruption**" permet d'activer ou désactiver l'interruption du module RTCC.

Enfin il existe 4 fonctions de conversion qui permettent de changer la date en differents types : type RTCC_DATE, type unsigned long ou chaîne de caractère.

3 - Le MRF89

Ce composant permet de communiquer par liaison hertzienne entre 2 ou plusieurs dispositifs.

Dans ce paragraphe on ne détaillera pas toutes les fonctions de la librairie.

3.1 - Configuration du MRF89

Pour utiliser ce composant il faut commencer par configurer la liaison SPI avec la fonction suivante :
void Lib_Temp_SPI_Config(unsigned char Num_SPI); // Numero SPI de 1 à 3

Puis utiliser suivre les étapes suivantes :

- Configurer une variable de type "**MRF89_DEFINE_PORT**" en indiquant pour chaque champ le nom du bit correspondant au câblage de votre carte.
- Appeler la fonction "**Lib_Temp_MRF89_Config**".

3.2 - Emission d'un message

Pour envoyer un message, il faut utiliser la fonction : "**Lib_Temp_MRF89_Ecriture_SPI_FIFO**", puis attendre la fin de la transmission avec la fonction :
"**Lib_Temp_MRF89_Attend_Fin_Transmission_Hertzienne**".

3.3 - Réception d'un message

Pour recevoir un message, il faut placer le MRF89 en mode réception avec la commande :
Lib_Temp_MRF89_Change_Mode((unsigned char)MRF89_MODE_RECEIVE);

Puis tester la fonction "**Lib_Temp_MRF89_Read_SPI_Config_Register**" qui retourne le nombre d'octets reçus. Si le nombre est nul, c'est qu'il n'y a pas eu de réception.

3.4 - Mise en basse consommation

Pour économiser l'énergie consommée par le MRF89, il faut le placer en mode sleep avec la commande suivante :
Lib_Temp_MRF89_Change_Mode(MRF89_MODE_SLEEP);